MakePython **ESP32 Development** Kit

alles in een doosje

Tam Hanna (Slowakije)

Moderne microcontrollers, zoals de ESP32, zijn zo krachtig dat ze in MicroPython geprogrammeerd kunnen worden. Dankzij de krachtige bibliotheken kunt u zo snel een afgerond project realiseren. Met de MakePython ESP32 Development Kit, enerzijds een (Engelstalig) leerboek en anderzijds een hardwarekit, presenteert de bekende Elektor-auteur Dogan Ibrahim nu een starterspakket dat aan de hand van praktijkvoorbeelden illustreert hoe u met MicroPython kunt werken.

We hoeven hier niet uit te leggen dat MicroPython niet de manier is om maximaal efficiënte software te maken. Maar moderne microcontrollers zoals de ESP32 kunnen het qua prestaties gemakkelijk opnemen tegen een 486. En daarom kan het, zeker voor kleine series, heel redelijk zijn om wat snelheid op te offeren en daarbij werk te besparen door een hogere programmeertaal te gebruiken.

Het boek

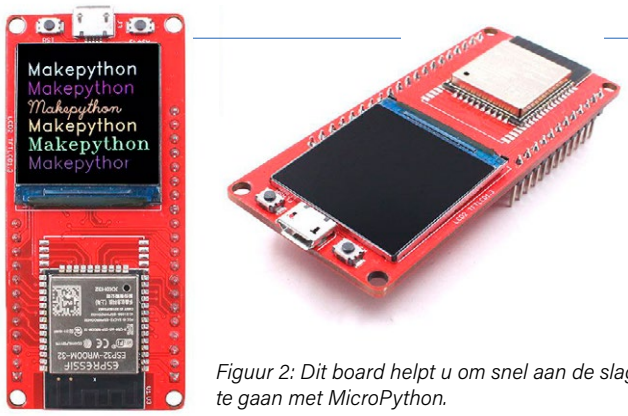
We beginnen met het Engelstalige leerboek: Het ziet er best flitsend uit, met een (imitatie van) een spiraalbinding.

De eigenlijke didactische structuur begint met een korte uitleg van de ESP32 als geheel. Een Elektor-lezer met uitgebreide of op zijn minst basiskennis over microcontrollers, vind hier een "snel" overzicht van de periferie die Espressif in deze controller levert. Als u geen ervaring hebt met microcontrollers, is de uitleg (op sommige punten) misschien te kort om volledig te begrijpen.

De installatie van de IDE's (Ibrahim introduceert zowel uPyCraft als Thonny, maar werkt verder bijna uitsluitend met Thonny) wordt in detail uitgelegd voor Windows, maar over Linux staat er weinig. Dan volgt een hoofdstuk dat de uitvoering van enkele "kleine" stukjes Python met Thonny illustreert, wie nog geen kennis heeft van de syntax van Python zal hier weinig van begrijpen. Maar de uitleg is meer dan voldoende om de basis van het werken met MicroPython en de IDE te begrijpen.

De eigenlijke presentatie van de 46 projecten in het boek gaat in de klassieke Dogan Ibrahim-stijl: als eerste stap presenteert de professor altijd de "te volbrengen missie", en dan volgt de code en uitleg over het bouwen van de hardware. Fans van compact coderen zullen zich misschien ergeren aan het feit dat de listings altijd beginnen met een standaard header van meer dan tien regels. Maar de voorbeelden (die niet te ingewikkeld zijn) laten heel duidelijk de essentiële aspecten van het werken met MicroPython zien.

Figuur 1: De kit. De plastic behuizing beschermt de componenten.



Figuur 2: Dit board helpt u om snel aan de slag te gaan met MicroPython.

Als u dit hebt gelezen, hoeft u niet meer bang te zijn dat u geen idee hebt hoe u bepaalde periferie aan de praat moet krijgen. Wat ik ook heel positief vind, is dat Ibrahim uitlegt hoe een "verprutste" kaart opnieuw met MicroPython-firmware kan worden geladen.

De ontwikkelkaart

De ontwikkelkaart is in combinatie met het boek verkrijgbaar in de kit in **figuur 1**, die ook geschikt is om mee op reis te nemen dankzij de behoorlijk stevige plastic behuizing. Maar laten we de eigenlijke hardware eens bekijken. De auteur heeft door praktische onderwijservaring geleerd dat vooral ontwikkelaars die niet bekend zijn met embedded systemen een veel prettiger leerervaring hebben als het ontwikkelplatform beschikt over een (liefst volledig grafisch) display.

Elektor lost dat slim op met de rode print die u ziet in **figuur 2**. Het zwarte gebied is geen zwart gat, maar één van die bekende SSD1306-OLED's met een resolutie van 128x64 pixels.

Mijn grootste kritiekpunt is hier niet te zien: De 2,54mm-headers moeten door de gebruiker worden ingesoldeerd. Dat is heel jammer, want als iemand de kit koopt in een winkel en hem meteen meeneemt op vakantie zit hij klem, tenzij hij ook een soldeerbout in de koffer heeft, of de telefoonwinkel om de hoek zo vriendelijk is om hier even mee te helpen.

Voor de rest is er niets aan te merken op de kaart. De gebruikte ESP32-variant is één van de grotere modellen en heeft genoeg geheugen, de MicroUSB-interface garandeert dat een "besturingskabel" wel te vinden is bij een oudere mobiele telefoon. En de kit bevat zelf ook een (heel korte) kabel van behoorlijke kwaliteit.

Een demo

Hoewel het leerboek voornamelijk gericht is op de behoeften van ontwikkelaars die werken met Windows, ga ik bij de volgende stappen gebruik maken van Ubuntu 20.04 LTS, alleen maar voor het gemak. De op [1] beschikbare IDE is een slechte keus, want op alle Ubuntu-versies later dan 16.04 geeft die een foutmelding zoals:

```
ImportError: /tmp/_MEIOhQKhz/libz.so.1: version
`ZLIB_1.2.9' not found (required by /usr/lib/x86_64-
linux-gnu/libpng16.so.16)
```

Thonny is dan een stuk "vriendelijker". Die is automatisch te installeren met het commando:

```
bash <(wget -O - https://thonny.org/installer-for-linux)
```

en kan dan worden gestart met:

```
/home/tamhan/apps/thonny/bin/thonny
```

Om Thonny te verwijderen, gebruiken we `/home/tamhan/apps/thonny/bin/uninstall`. Alles wat we nog hoeven te doen bij de eerste start is de taalkeuze te bevestigen.

De volgende stap is het verbinden van het evaluatieboard met een werkstation. Gelukkig heeft Elektor er aan gedacht om de kaart te leveren met een voorgeconfigureerde MicroPython-runtime. Het is heel handig, dat de kaart ook meteen het display inschakelt, zodat we kunnen zien dat het werkt. Als converter wordt een Silicon Labs CP210x UART bridge chip gebruikt.

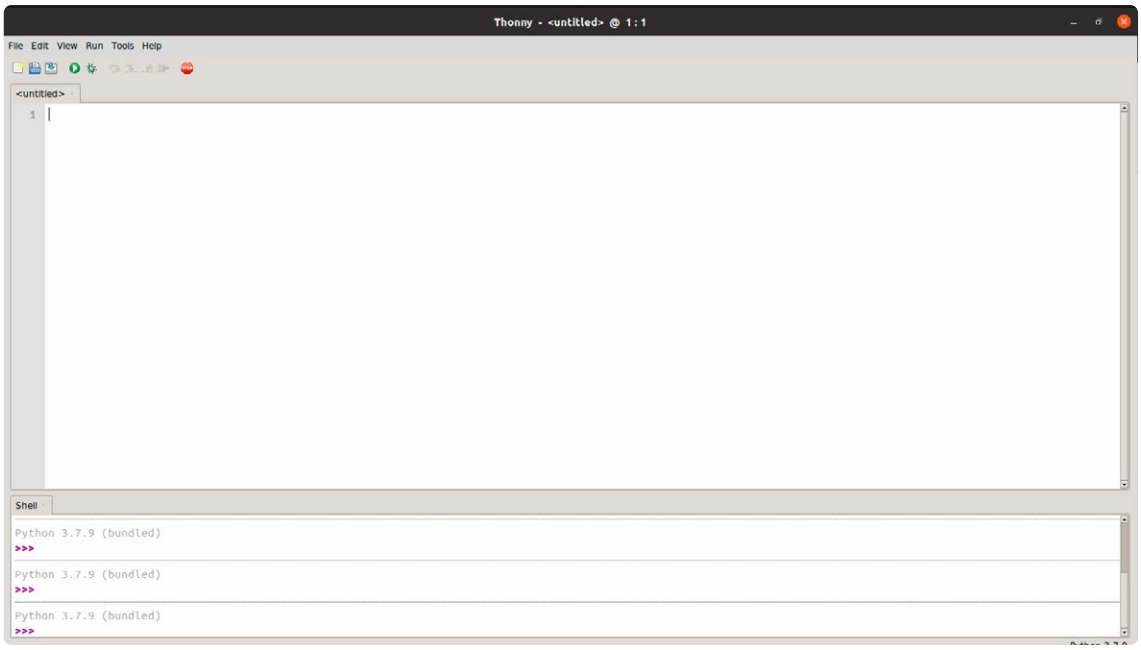
U kunt nu overschakelen naar Thonny (**figuur 3**). Klik in het keuzemenu op de Python-versie die rechtsonder verschijnt en kies *MicroPython (ESP32)*. Thonny gaat dan automatisch op zoek naar ESP32-kaarten. In mijn geval werd de aangesloten kaart automatisch gedetecteerd.

Het is de vraag of het nodig is om Thonny te draaien met superuser-privileges. Mijn eigen user-account is lid van de groep *plugdev*, dus ik kon Thonny gewoon draaien op mijn account en had toegang tot de ESP32.

Een eerste welkom

Het wordt nu tijd om het OLED-display tot leven te brengen. Open de URL: *www.makerfabs.com/makepython-esp32-starter-kit.html*. Download het archief *MakePython ESP32 Lessons Source Code*. Werken met RAR is niet zo prettig onder Linux. Pak het archief uit naar een gemakkelijk toegankelijke locatie. Voor de volgende stappen hebben we vooral het bestand *ssd1306.py* nodig, dat de driver voor het display bevat.

Klik om te beginnen op *File -> Open* en kies dan de optie *This Computer*. Navigeer dan naar het bestand en laad het in de editor.



Figuur 3: Screenshot van Thonny.

Dan kiezen we *File -> Save* en de optie *MicroPython Board*.

Als Thonny daarbij begint te mopperen dat het backend "busy" is, klik dan op het rode stop-pictogram in de toolbar om hem te stoppen. Sla het bestand dan op in het bestandssysteem onder de bekende naam (alleen als u de originele firmware hebt veranderd). Zo niet, dan staat de file al klaar.

Zoals gebruikelijk bij MicroPython moeten we eerst enkele bibliotheken installeren en ook constanten definiëren met extra informatie over de specificaties van het display:

```
import machine
import ssd1306
import time
WIDTH = const(128)
HEIGHT = const(64)
```

Voor de eigenlijke communicatie met het display gebruiken we I2C (dat is niet heel gebruikelijk, meestal zou je SPI verwachten). We hebben daarvoor de software-I2C-class nodig:

```
pscl = machine.Pin(5, machine.Pin.OUT)
psda = machine.Pin(4, machine.Pin.OUT)
i2c = machine.I2C(scl=pscl, sda=psda)
oled = ssd1306.SSD1306_I2C(WIDTH, HEIGHT, i2c)
```

Tenslotte sturen we tekst naar het display op de volgende manier:

```
while True:
    oled.fill(0)
    oled.text('Hello World!',10,0)
    oled.show()
    time.sleep(1)
```

Nu kunt u op het play-pictogram in de IDE drukken en genieten van de tekst op het display van de ontwikkelkaart.

MQTT opzetten

Nu willen we een beetje voorbij de "gewone" projecten in de kit gaan en een geavanceerde taak uitvoeren, om te laten zien dat de kit geschikt is voor moeilijker taken. Om precies te zijn, willen we het bekende MQTT-protocol gaan gebruiken. Het wordt vooral interessant omdat we willen werken met een gloednieuwe versie van de MQTT-broker Mosquitto. Bij [2] vindt u een (aanbevelenswaardige) beschrijving van de problemen die kunnen optreden met de parsers (die om veiligheidsredenen steeds strikter worden geconfigureerd) in interactie met de MicroPython-drivers.

Nu we het toch over MicroPython-drivers hebben: er zijn intussen twee concurrerende implementaties van MQTT voor MicroPython. We gaan hier gebruik maken van de eenvoudigste, die te vinden is op [3]. Download de file:

https://github.com/micropython/micropython-lib/blob/master/micropython/umqtt.simple/umqtt/simple.py

En sla die op als *simple.py* op de computer.

Nu moeten we de WiFi-interface van de kaart in station-modus zetten op de volgende manier:

```
. . .
wlan=network.WLAN(network.STA_IF)
```

Voor het opzetten van de eigenlijke verbinding gebruiken we een methode gebouwd volgens het volgende ontwerp, deze is grotendeels afkomstig van het projectskelet van MakerFabs en implementeert een countdown-teller inclusief het maken van de verbinding.

Listing 1.

```
def connectWiFi():
  i=0
  wlan.active(True)
  wlan.disconnect()
  wlan.connect("ssid","pass")
  while(wlan.ifconfig()[0]=='0.0.0.0'):
    i=i+1
    oled.fill(0)
    oled.text('connecting WiFi',0,16)
    oled.text('Countdown:'+str(20-i)+'s',0,48)
    oled.show()
    time.sleep(1)
    if(i>20):
      break
  oled.fill(0)
  oled.text('Makerfabs',25,0)
  oled.text('MakePython ESP32',0,32)
  if(i<20):
    oled.text('WIFI connected',0,16)
  else:
    oled.text('NOT connected!',0,16)
  oled.show()
  time.sleep(3)
  return True
```

Natuurlijk moet u de strings voor `wlan.connect` aanpassen aan uw lokale netwerk (zie **listing 1**).

Het maken van de eigenlijke testverbinding is dan heel eenvoudig:

```
connectWiFi()
while True:
    time.sleep(1)
```

Vervolgens willen we een Mosquitto-server. Mosquitto is niet de snelste MQTT-broker, maar hij is open source, gratis en zeer strikt in zijn MQTT-implementatie.

Omdat hij zo veel gebruikt wordt, is er ook een gebruiksklaar image in de Docker Hub, waarmee u de server kunt draaien zonder ingrijpende herconfiguratie van de hostcomputer.

Voor een eerste poging volstaat het volgende commando:

```
docker run -it -p 1883:1883 -p 9001:9001 -v mosquitto.conf:/mosquitto/config/mosquitto.conf eclipse-mosquitto
```

De parameters `-p 1883:1883` en `-p 9001:9001` zijn belangrijk. Elke Docker-container heeft immers beschikking over alle TCP/IP-poorten van het systeem. Elk van de parameters verbindt één van deze poorten met de netwerkkaart van het hostcomputer. Met de parameter `-v mosquitto.conf:/mosquitto/config/mosquitto.conf` integreren we ook een configuratiefile.

Op dit moment is het een goed idee om een eerste download-poging te doen met een bestaande Internetverbinding. Docker zal normaal gesproken verbinden met de hub, de vereiste componenten downloaden en stoppen met de volgende foutmelding:

```
docker: Error response from daemon: source /var/lib/docker/aufs/mnt/a22b9e557c37d99eb71f17e7bc6d38df6e7677d09225376d416612adf0977ccd/mosquitto/config/mosquitto.conf is not directory.
```

De oorzaak van die fout is dat er nog geen bestand *mosquitto.conf* staat in de hostcomputer.

Die file kunnen we gewoon zelf maken met `gedit`:

```
(base) tamhan@tamhan-thinkpad:~$ gedit mosquitto.conf
```

De overgang van Mosquitto versie 1 naar versie 2 is gepaard gegaan met een heftige verstrakking van de veiligheidsconfiguratie. Een Mosquitto 1.X (gestart met default parameters) aanvaardde server-verzoeken van alle klanten, Mosquitto 2 weigert dat. Maar als u de volgende regels in het configuratiebestand zet, werkt alles weer net zo gemakkelijk (en onveilig) als vroeger:

```
listener 1883
allow_anonymous true
```

De eigenlijke uitvoering kan dan worden gestart met het onderstaande commando. Let op: de `-v` parameter heeft altijd een volledig pad nodig, daarom gebruiken we het `pwd`-commando en "bouwen" het commando op met shell-functionaliteit. `pwd`, staat trouwens voor Print Working Directory, **figuur 4** laat zien hoe het werkt:

```
(base) tamhan@tamhan-thinkpad:~$ docker run -it -p 1883:1883 -p 9001:9001 -v $(pwd)/mosquitto.conf:/mosquitto/config/mosquitto.conf eclipse-mosquitto
. . .
```

Als de Docker-container de start bevestigt met `1658673183: mosquitto version 2.0.14 running`, hebben we een server draaien.

```
tamhan@tamhan-thinkpad: ~/Desktop/Stuff
Support: command not found
(base) tamhan@tamhan-thinkpad:~$ pwd
/home/tamhan
(base) tamhan@tamhan-thinkpad:~$ cd Desktop/Stuff/
(base) tamhan@tamhan-thinkpad:~/Desktop/Stuff$ pwd
/home/tamhan/Desktop/Stuff
(base) tamhan@tamhan-thinkpad:~/Desktop/Stuff$
```

Figuur 4: Het commando `pwd` geeft de huidige werkdirectory van de shell.

Initialiseren van de MQTT-software op de ESP32

Om te beginnen hebben we enkele constanten nodig om te werken met MQTT. Pas het mee te geven IP-adres in de serverstring aan voor uw lokale situatie:

```
SERVER = "192.168.178.146"
CLIENT_ID = ubinascii.hexlify(machine.unique_id())
TOPIC = b"led"
state = 0
```

We hebben ook een callback-functie nodig die de MQTT-driver altijd aanroept als er een bericht is ontvangen:

```
def sub_cb(topic, msg):
    global state
    print((topic, msg))
    . . .
```

Het eigenlijke opzetten van de MQTT-verbinding is dan relatief eenvoudig:

```
connectWiFi()
c = MQTTClient(CLIENT_ID, SERVER, keepalive=30)
c.set_callback(sub_cb)
c.connect()
c.subscribe(TOPIC)
print("Connected to %s, subscribed to %s topic" % (SERVER,
TOPIC))

while True:
    c.wait_msg()
```

De aanroep `c.subscribe` zorgt dat de MQTT-driver de server vertelt, welke kanalen we willen ontvangen. Het is ook belangrijk om periodiek de methode `c.wait_msg()` aan te roepen om de MQTT-server CPU-tijd te geven voor het verwerken van de informatie die binnenkomt of wordt verzonden. U ziet de volgende output `Connected to 192.168.178.146, subscribed to b'led' topic`.
Nu willen we één van de beschikbare LED's verbinden met GPIO-pen 12. Natuurlijk via een serieweerstand, we gaan ervan uit dat u voldoende kennis hebt van elektronica. Voor de initialisatie van de GPIO-poort is alleen gewone ESP32-code nodig:

```
led = Pin(12, Pin.OUT, value=1)
```

Om binnenkomende berichten te verwerken, moeten we de callback uitbreiden door de string in `topic` te combineren met de constanten voor de diverse commando's:

```
def sub_cb(topic, msg):
    global state
    print((topic, msg))
    if msg == b"on":
        led.value(1)
        state = 1
    elif msg == b"off":
        led.value(0)
        state = 0
    elif msg == b"toggle":
        led.value(state)
```

Voor de eigenlijke test gebruiken we het commando *mosquitto_pub*: Dit is een Linux-tool om commando's rechtstreeks naar een MQTT-server te sturen. De volgende commando's schakelen de LED aan en uit:

```
(base) tamhan@tamhan-thinkpad:~$ mosquitto_pub -t led
-m 'on'
(base) tamhan@tamhan-thinkpad:~$ mosquitto_pub -t led
-m 'off'
```

Als de ESP32 en de server in hetzelfde netwerk zitten, kunt u nu de LED aan- en uitschakelen.

Conclusie en vooruitblik

De MakePython ESP32 Development Kit is een verbluffend compacte gereedschapskist, waarmee Python-ontwikkelaars en/of elektronici snel die twee werelden kunnen verenigen. Vooral omdat alles zo handig verpakt en klaar voor gebruik is, is het een doosje dat ik van harte kan aanbevelen.

220429-03

Related Products

- **MakePython ESP32 Development Kit (SKU 20137)**
  www.elektor.nl/20137

WEBLINKS

[1] uPyCraft IDE: https://github.com/DFRobot/uPyCraft
[2] umqtt.simple en mosquitto 2.0.12: https://github.com/micropython/micropython-lib/issues/445
[3] umqtt.simple : eenvoudige MQTT-client voor MicroPython:
https://github.com/micropython/micropython-lib/tree/master/micropython/umqtt.simple